Kritikus Beágyazott Rendszerek

Megbízhatósági modellezés

Gyakorlat

Készítette: Vörös András

Utolsó módosítás: 2015-09-22

Verzió: 1.0

Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

1 Bevezető

A megbízhatósági modellezés egyre fontosabb szerepet kap napjaink informatikai infrastruktúráinak tervezésében. Ahogy a szolgáltatásokra egyre nagyobb feladat hárul, kiesésük is egyre költségesebb a cégeknek. Az alábbi táblázatban egy tanulmány adatai szerepelnek 2003-ból, melyben jól látható, hogy már évekkel ezelőtti is komoly költségekkel járt egyes iparágakban a szolgáltatások kiesése:

Esettanulmány	**Éves bevétel**	**Kiesés költsége**	**Költség/óra**
Energiaszektor	6.75 milliárd $	4.3 millió $	1624 $
„High tech”	1.3 milliárd $	10.2 millió $	4,167 $
Egészségügy	44 milliárd $	74.6 millió $	96,632 $
Utazás	850 millió $	2.4 millió $	38,710 $
Pénzügyi szektor (USA)	4.0 milliárd $	10.6 millió $	28,342 $
Pénzügyi szektor (Európa)	1.2 milliárd $	379,000 $	1573 $

A következő táblázat frissebb adatokat tartalmaz, egy 2009-es felmérés iparágak és tevékenységek szerint lebontva vizsgálta az üzleti tevékenységek veszteségeit a rendszerek megbízhatatlansága miatt:

Iparág	**üzleti tevékenység**	**átlagos óránkénti kiesés költsége**
pénzügyi szektor	ügynöki, közvetítői	6.45 millió $
pénzügyi szektor	hitelkártya, pénzügyi tranzakciók	2.6 millió $
média	pay-per-view	150 000 $
kiskereskedelem	TV shopping	113 000 $
utazás	repülőjegy foglalás	90 000 $

Hogy ezek a költségek tervezhetőek és előre jósolhatóak legyenek, szükségünk van a megbízhatósági modellezésre, és a mögöttes matematikai apparátusra.

A segédlet célja, hogy megalapozza a gyakorlatot a modellezési alapismeretek felfrissítésével, továbbá hogy a mérés során segítséget nyújtson az eszközök használatához.

2 Modellezési formalizmusok

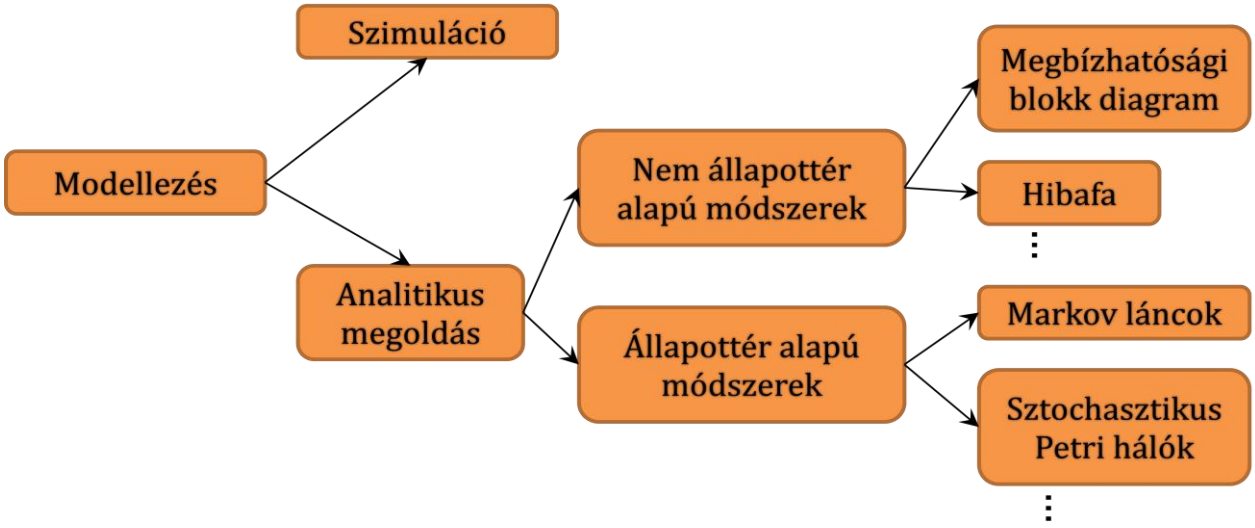
A megbízhatóság számítására többféle lehetőségünk van:

- szimuláció
- analitikus megoldás

A szimuláció előnye, hogy bármilyen modellre alkalmazható, nincsenek olyan szoros megkötések (eloszlások, modellezhető viselkedések), mint ami az analitikus megoldhatóságot jellemzi. Nagy hátránya azonban, hogy csak korlátozott az így nyert információ, sok esetben nem eldönthető, hogy elég esetre, időre futtattuk-e a szimulációt. Az így kapott eredményeket óvatosan kell kezelni.

Az analitikus megoldás előnye, hogy pontos eredményt ad, ellenben sok modellre nem alkalmazható, különösen a dinamikus modelleknél vannak korlátai.

A különböző lehetőségeket a következő ábra szemlélteti:



1. ábra: Megbízhatósági modellek

A továbbiakban végignézzük a gyakorlat során a megbízhatósági modellezéshez használt eszközöket, példák segítségével megvizsgáljuk a modellezési és analízis lehetőségeket. A gyakorlat során hasonló feladatokat kell majd megoldani.

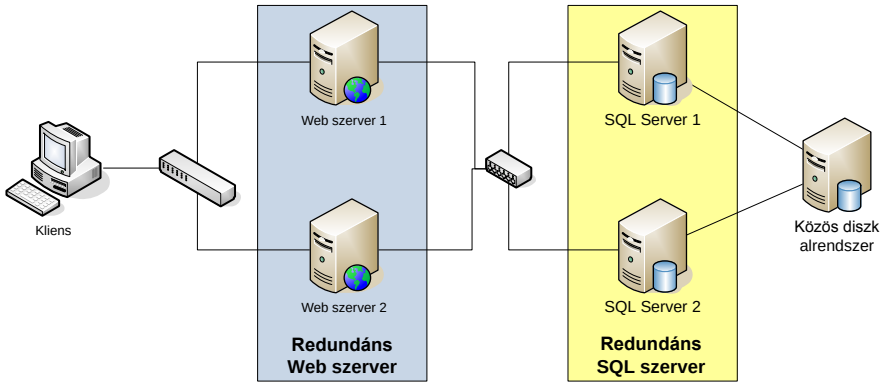
3 Nem állapottér alapú modellek

Az alábbi fejezetben a **SHARPE** eszköz segítségével áttekintjük a hibafa-rajzolást, és a hibafák által nyújtott analízis lehetőségeket, majd áttérünk a másik gyakran használt nem állapottér alapú analízisre, a megbízhatósági blokk diagramokra (RBD), és megnézzük a mintapélda RBD modelljét.

3.1 Infrastruktúra

A következőkben egy egyszerű számítógépes infrastruktúra hibafáját nézzük meg. A példa infrastruktúra egy egyszerű webes és egyéb szolgáltatásokat nyújtó hálózat leegyszerűsített modellje.

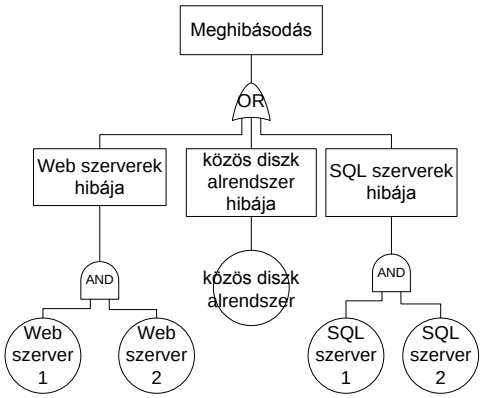
Az infrastruktúra egy webszerver fürtből, egy SQL szerver fürtből és az adatbázis szerverek által használt közös diszk alrendszerből áll. Ez a webszerverek és az SQL szerverek esetén is redundáns megoldás növeli a megbízhatóságot, hiszen ha kiesik az egyik szerver, attól még a teljes szolgáltatás nem esik ki. A példa során megnézzük, hogy mi vezethet az infrastruktúra leállásához.



2. ábra: webes infrastruktúra

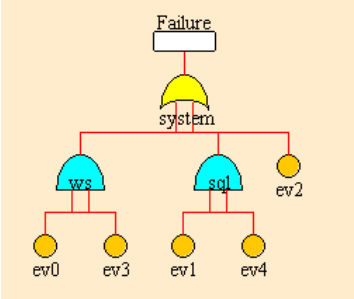
3.2 Hibafa rajzolás

A feladat során először felvesszük a legfelsőbb szintű eseményt: ez lesz a teljes rendszerhiba. A rendszer működéséhez legalább egy web szervernek, egy SQL szervernek és a közös diszk alrendszernek működnie kell.



3. ábra: webes infrastruktúra hibafája

A fenti hibafa **SHARPE** eszközben megrajzolt változata (*ev* események a komponensek meghibásodásai):



4. ábra: webes rendszer SHARPE hibafája

Az ábrán látható, hogy a SPOF (Single Point of Failure, egyszeres meghibásodási pont) a közös diszk alrendszer, hiszen a meghibásodása már önmagában elég a rendszer leállásához.

Megjegyzések a **SHARPE** eszköz használatához:

- Egy kaput, ha már letettünk, akkor csak törölni tudjuk, sem a nevét, sem a bemenő lábainak számát nem tudjuk változtatni (a legfelső szintű kapuknál érdemes előre eltervezni, hogy hány lábra is lesz szükség)
- A kapuk neve sem „*and*”, sem „*or*” (sem „*AND*” és „*OR*”) nem lehet, és ha már adtunk nevet a kapuknak, a nevüket megváltoztatni nem tudjuk, tehát érdemes nem rögtön a legelején elrontani egy rossz névvel
- Az elemek neveiben sem szóköz, sem kötőjel, sem ékezet vagy bármilyen speciális karakter nem szerepelhet
- Mielőtt átlépünk egy másik eseményre a szerkesztő ablakban, nyomjuk meg a *Validate* gombot! Az értékeket akkor fogadja el, ha narancssárga lesz az esemény.
- Tizedespontot kell használni és nem tizedesvesszőt.
- Új kapu az eseményre vagy csomópontra való kattintással adható hozzá.

3.3 Megbízhatósági mérés:

A komponensek megbízhatóságát λ paraméterű exponenciális eloszlással szoktuk jellemezni, ahol a meghibásodási idő várható értéke 1/ λ.

Rendelkezzenek a komponensek a következő megbízhatósági adatokkal:

komponens	λ
webszerver	0.05
SQL szerver	0.01
közös diszk	0.2

Ebből a **SHARPE** segítségével kiszámolhatjuk a megbízhatósági görbét:

Az *Analysis Editor →Analysis* menüpontját kiválasztva jutunk el analízis ablakhoz.

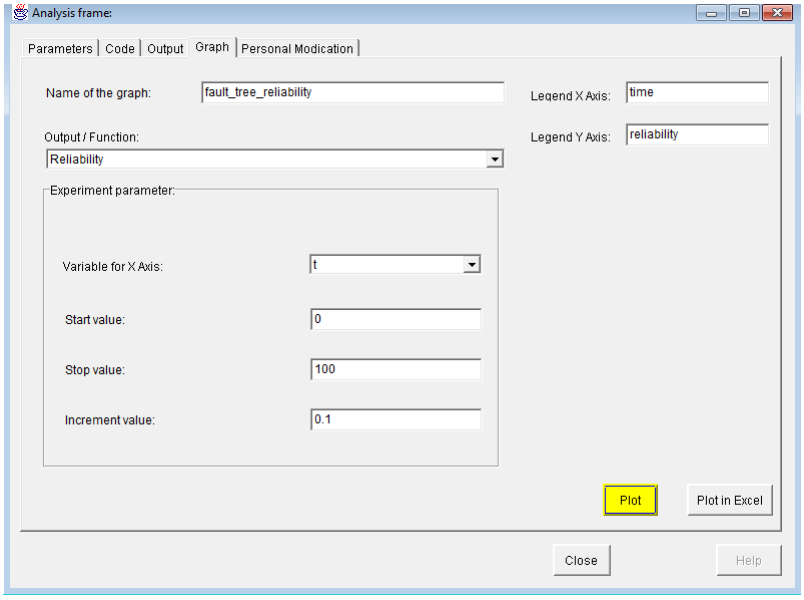
Ha az analízis ablakban a *Parameters* fülre kattintunk, akkor a következő értékeket számolhatjuk ki a programmal:

- *Reliability*, azaz megbízhatóság (egy adott időpillanat)
- *Unreliability*
- *Mean Time to Failure* (MTTF), azaz a meghibásodás várható ideje
- *Variance*, azaz szórásnégyzet

Ha az előbb látott modell esetén megkérdezzük a programtól, hogy mi a megbízhatóság várható értéke a *t* = 10 időpillanatban, akkor a következő kimenetet kapjuk:

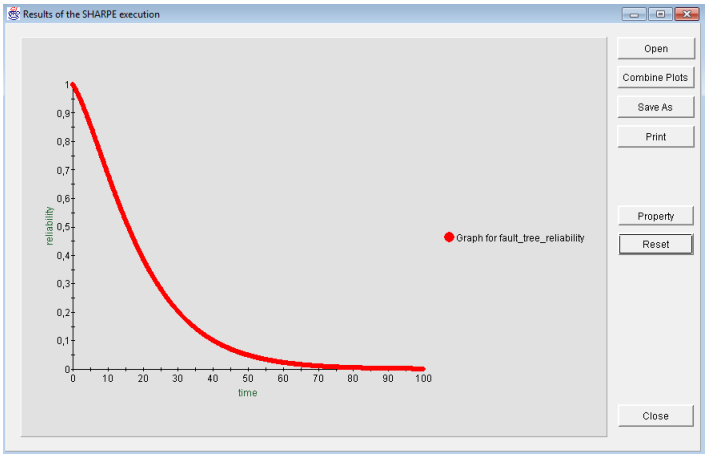
```
**********************************************************************
 *********  Outputs asked for the model: ft **************
 Reliability at time 10
 Reliability:   1.13347087e-001
```

Ha az analízis ablakban a *Graph* fülre kattintunk, akkor a következő ablakhoz jutunk:



5. ábra: Hibafa analízis beállítások

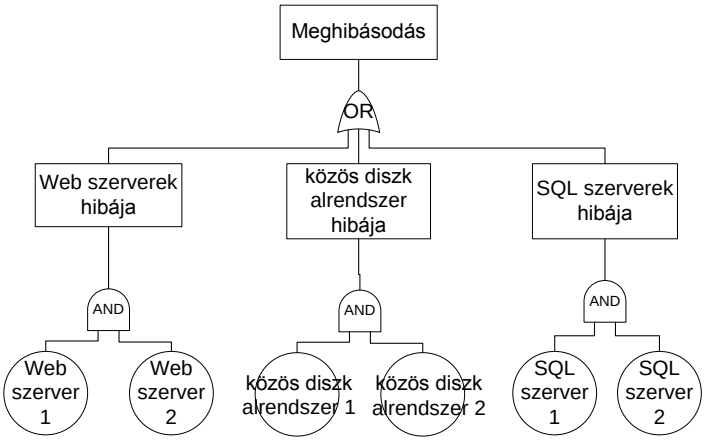
Fontos a paraméterek megfelelő kitöltése, nem érdemes sokkal tovább futtatni a megjelenítést, mint ahol még releváns eredményeket láthatunk. A paramétereket megfelelően beállítva, majd a *Plot* gombra kattintva a következő ábrát kapjuk.



6. ábra: Hibafa megbízhatósága

Az így kapott diagramot elmenthetjük a program segítségével, ha a *Save As* gombra kattintunk. Ez akkor praktikus, ha később majd több függvényt szeretnénk egy diagramon megjeleníteni.

Vizsgáljuk meg, hogy ha a közös diszk alrendszert is redundánssá tesszük, akkor hogyan változik meg a rendszer megbízhatósága. Ehhez a diszk alrendszernek létrehozunk egy AND kaput, mellyel összekötjük a különálló diszk alrendszer eseményeket:

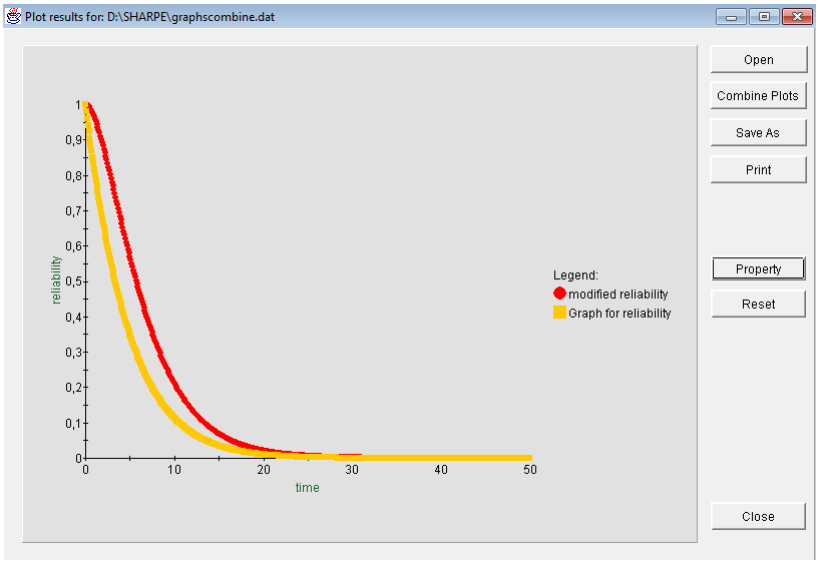


7. ábra: Módosított webes infrastruktúra hibafája

Ekkor a korábban is említett *Reliability* menüpont a következő kimenetet produkálja:

```
**************************************************************
 *********  Outputs asked for the model: ft **************
 Reliability at time 10
 Reliability:   2.11354313e-001
----------------------------------------
```

Ha meg szeretnénk jeleníteni a korábbi és a módosított megbízhatósági értékeket egy ábrán, akkor először megjelenítjük a módosított modell megbízhatósági függvényét a megfelelő paraméterek beállítása után a *Plot* gombra kattintva. Fontos, hogy a két ábra felbontásának egyeznie kell, tehát ugyanaddig a pontig kell futtatni a megjelenítést, különben a **SHARPE** hibát fog jelezni. Ha megjelenítettük a módosított modell megbízhatósági függvényét, akkor a *Combine Plots* gombra kattintva megnyithatjuk mellé a korábban elmentett régi megbízhatósági függvényünket, amit a program utána egy közös ablakban megjelenít:



8. ábra: Modellek összevetése megbízhatóság szempontjából

Az ábrán sárgával jelöltük a rendszer megbízhatóságát, és pirossal a módosított modell alapján számolt megbízhatósági függvényt.

Jól látható, hogy növekedett a rendszer megbízhatósága.

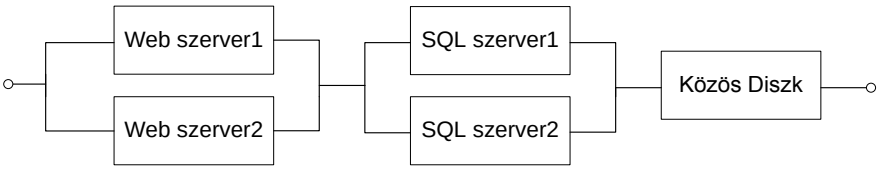
3.4 Megbízhatósági Blokk Diagram (Reliability Block Diagram, RBD)

A **SHARPE** eszközben elkészíthetjük a rendszer megbízhatósági blokk diagramját, azaz RBD-jét. Ehhez érdemes új *project*-et létrehozni, de a **SHARPE** lehetőséget nyújt akár arra is, hogy hierarchikusan kombináljuk a különböző modellezési formalizmusokat, azaz RBD-vel leírhatjuk az egyik komponensen (hibafa komponens) bekövetkező eseményt. Mi most a példában egy külön project-et hozunk létre.

Az RBD-k sorosan és párhuzamosan kötött modellelemekből állnak. Az RBD-kben azzal a feltételezéssel élünk, hogy meghibásodni csak komponens tud, összeköttetés nem, azaz a modellelemek közötti összeköttetések csak a hibaterjedés logikai leírására szolgálnak. Továbbá a komponensek meghibásodása független.

Ha n darab sorosan kötött elem megbízhatóságai $R_1 \ldots R_n$, akkor az eredő közös megbízhatóság: $R_e = \prod R_n$, ha párhuzamosan kötjük őket, akkor $1 - R_e = \prod(1 - R_n)$.

Ezek alapján a fenti példarendszer RBD-je a következőképpen néz ki:



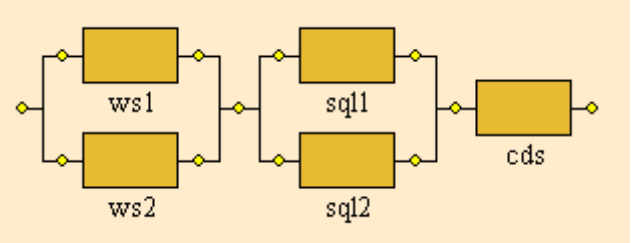
9. ábra: Webes rendszer RBD-je

A modellek SHARPE eszközben történő elkészítésekor:

- a *Series* gombra kattintva, utána pedig az egyik modellelemre kattintva tudjuk az adott modellelemet soros komponensekre bontani
- a *Parallel* gombra kattintva, utána pedig az egyik modellelemre kattintva tudjuk az adott modellelemet párhuzamos komponensekre bontani
- mindkét esetben meg kell adni, hogy hány komponens jöjjön létre

- a *Parallel* esetben megadhatjuk az „*Identical to the current block*” checkbox-ot bejelölve, hogy az új elemek ugyanolyan típusúak legyenek, mint a kiinduló elem. Esetünkben, mivel mindkét webszerver és sql szerver ugyanolyan típusú, alkalmazhatjuk.
- kiinduló állapotban egy blokk van, és ezt bontjuk tovább, ezért figyelni kell, hogy ha a modellünk alapvetően soros viselkedésű, akkor először soros modellelemekre bontsuk, ha párhuzamos viselkedésű, akkor párhuzamos elemekre

A **SHARPE** eszközben elkészített modell a következőképpen fog kinézni:



10. ábra:Webes rendszer SHARPE RBD-je

Ha erre a modellre lefuttatjuk az analíziseket a fentebb (hibafáknál) megadott paraméterekkel, akkor ugyanazokat az eredményeket fogjuk kapni.